Bab 2. Kontrol Alur, *Method* & *Array*

OBJEKTIF:

1. Mahasiswa mampu belajar Kontrol Alur, *Method* & *Array* seperti mengetahui *Statement* `if`, *Statement* `if-else`, *Statement* `if-else if`, *Statement* `switch`, dan *Statement* `if` bersarang.
2. Mahasiswa mampu menggunakan struktur kondisi dan perulangan dengan baik.
3. Mahasiswa mampu memahami tentang pemakaian *method*.
4. Mahasiswa mampu memahami penggunaan *array*.

2.1 Struktur Kondisi

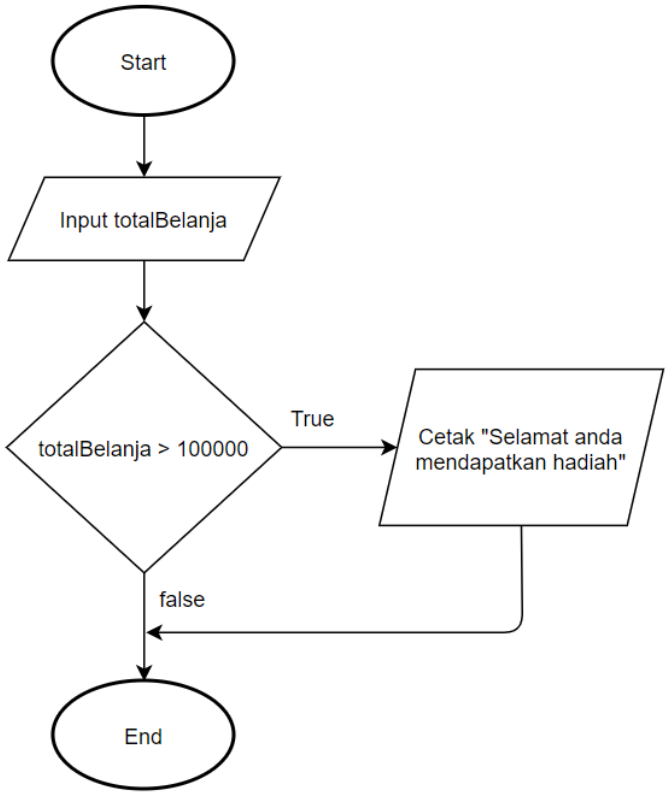
Dalam membuat program, seringkali kita ingin membuat alur program berjalan lebih dinamis. Tidak hanya berjalan sekuensial, tetapi alur berjalannya program bisa menyesuaikan dengan percabangan kondisi yang berada dalam struktur kode program.

2.1.1 *Statement* `if`

Statement `if` merupakan bentuk sederhana dari percabangan. *Statement* `if` ini hanya memiliki satu pilihan, artinya pilihan di dalam `if` akan dieksekusi ketika kondisi bernilai `true`. Jika *statement* bernilai `false`, maka program tidak akan melakukan apapun dan akan melanjutkan eksekusi perintah selanjutnya. Berikut merupakan bentuk umum dari *statement* `if`.

```
if(ekspresiBoolean)
{
    statement
    statement
    ...
}
```

Dengan menggunakan percabangan `if` saja sudah membuat alur program kita pelan-pelan berjalan secara dinamis dan tidak sekuensial lagi. Sebagai contoh, misal terdapat sebuah toko buku yang akan memberikan sebuah hadiah kepada pembeli jika nominal pembelian diatas $Rp 100.000$. *Flowchart* dari program tersebut adalah sebagai berikut.



Pada diagram *flowchart* diatas, akan dilakukan *input* nilai variabel `totalBelanja`. Selanjutnya akan dicek nilai dari variabel `totalBelanja` apakah lebih besar dari $Rp 100000$ atau tidak. Apabila nilai dari variabel `totalBelanja` lebih dari $Rp 100000$, maka program akan bernilai `true`. Selanjutnya program akan mencetak `Selamat anda mendapatkan hadiah`. Sedangkan jika nilai variabel `totalBelanja` kurang dari $Rp 100000$, maka program akan bernilai `false` dan tidak akan mengeksekusi apapun. Berikut merupakan kode program dari pengecekan variabel `totalBelanja`.

Program (Hadiah.java)

```
import java.util.Scanner;

public class Hadiah
{
    public static void main(String[] args)
    {
        // Membuat variabel totalBelanja dan scanner
        int totalBelanja;
        Scanner scan = new Scanner(System.in);

        // Memasukkan nilai totalBelanja
        System.out.print("Total Belanja : Rp ");
        totalBelanja = scan.nextInt();

        // Pengecekan apakah totalBelanja diatas 100000 atau tidak
        if (totalBelanja > 100000)
        {
            System.out.println("Selamat anda mendapatkan hadiah");
        }
    }
}
```

Output Program (Hadiah.java)

```
Total Belanja : Rp 120000
Selamat anda mendapatkan hadiah
```

Pada contoh program diatas, nilai variabel `totalBelanja` dimisalkan sebesar $Rp120000$ maka *statement* `if` bernilai `true`. Oleh karena itu, program akan otomatis menjalankan perintah mencetak `Selamat anda mendapatkan hadiah`. Apabila nilai variabel `totalBelanja` yang dimasukkan adalah dibawah $Rp 100000$ maka *statement* `if` bernilai `false` dan program tidak akan mengeksekusi perintah apapun.

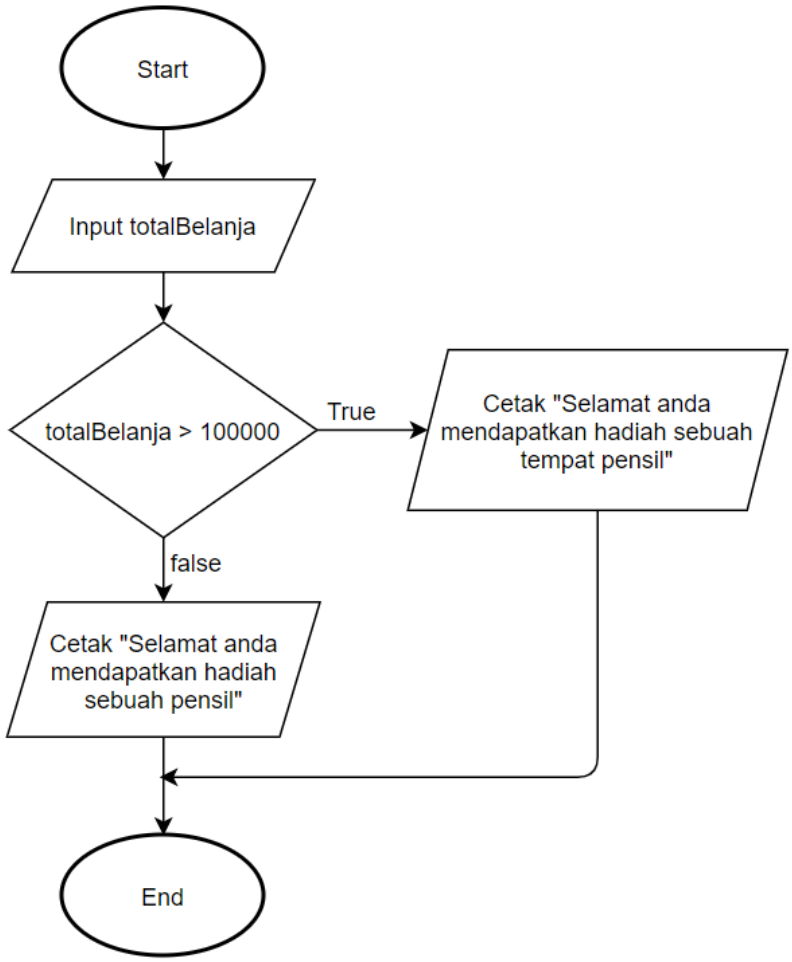
2.1.2 *Statement* `if-else`

Dalam struktur percabangan, nilai dari kondisi tidak hanya `true` tetapi terdapat nilai lain yaitu `false`. Kondisi `false` merupakan kondisi dimana nilai dari variabel yang diuji bernilai `false` atau salah. Untuk menangani kondisi `false` kita bisa menggunakan *statement* `else`. *Statement* `else` digunakan untuk mengeksekusi perintah ketika variabel yang diuji bernilai `false`.

Kita bisa menggabungkan *statement* `if` dan `else` untuk membuat alur program kita berjalan lebih dinamis. Berikut merupakan bentuk umum dari *statement* `if-else`.

```
if (ekspresiBoolean)
{
    statement
    statement
    ...
}
else
{
    statement
    statement
    ...
}
```

Kita dapat mengambil contoh dengan mengembangkan program pada contoh sebelumnya. Toko buku akan memberikan sebuah hadiah berupa tempat pensil kepada pembeli jika nominal pembelian diatas $Rp 100.000$. Selain itu, toko buku akan memberikan hadiah berupa sebuah pensil. *Flowchart* dari program tersebut adalah sebagai berikut.



Pada diagram *flowchart* diatas, akan dilakukan *input* nilai variabel `totalBelanja`. Selanjutnya akan dicek nilai dari variabel `totalBelanja` apakah lebih besar dari $Rp 100000$ atau tidak. Apabila nilai dari variabel `totalBelanja` lebih dari $Rp 100000$, maka program akan bernilai `true`. Selanjutnya program akan mencetak `Selamat anda mendapatkan hadiah sebuah tempat pensil`. Sedangkan jika nilai variabel `totalBelanja` kurang dari $Rp 100000$, maka program akan bernilai `false` dan akan mencetak `Selamat anda mendapatkan sebuah pensil`. Berikut merupakan kode program dari pengecekan variabel `totalBelanja`.

Program (Hadiah.java)

```
import java.util.Scanner;

public class Hadiah
{
    public static void main(String[] args)
    {
        // Membuat variabel totalBelanja dan scanner
        int totalBelanja;
        Scanner scan = new Scanner(System.in);

        // Memasukkan nilai totalBelanja
        System.out.print("Total Belanja : Rp ");
        totalBelanja = scan.nextInt();

        // Pengecekan apakah totalBelanja diatas 100000 atau tidak
        if (totalBelanja > 100000)
        {
                System.out.println("Selamat anda mendapatkan hadiah sebuah tempat
pensil");
```

```
        }
        else
        {
            System.out.println("Selamat anda mendapatkan hadiah sebuah pensil");
        }
    }
}
```

Output Program (Hadiah.java)

```
Total Belanja : Rp 20000
Selamat anda mendapatkan hadiah sebuah pensil
```

Pada contoh program diatas, nilai variabel `totalBelanja` dimisalkan sebesar $Rp20000$ maka *statement* `if` bernilai `false` dan akan langsung mengeksekusi perintah di dalam `else`. Oleh karena itu, program akan otomatis menjalankan perintah mencetak `Selamat anda mendapatkan hadiah sebuah pensil`. Apabila nilai variabel `totalBelanja` yang dimasukkan adalah diatas $Rp 100000$ maka *statement* `if` bernilai `true` dan program akan mencetak `Selamat anda mendapatkan hadiah sebuah tempat pensil`.

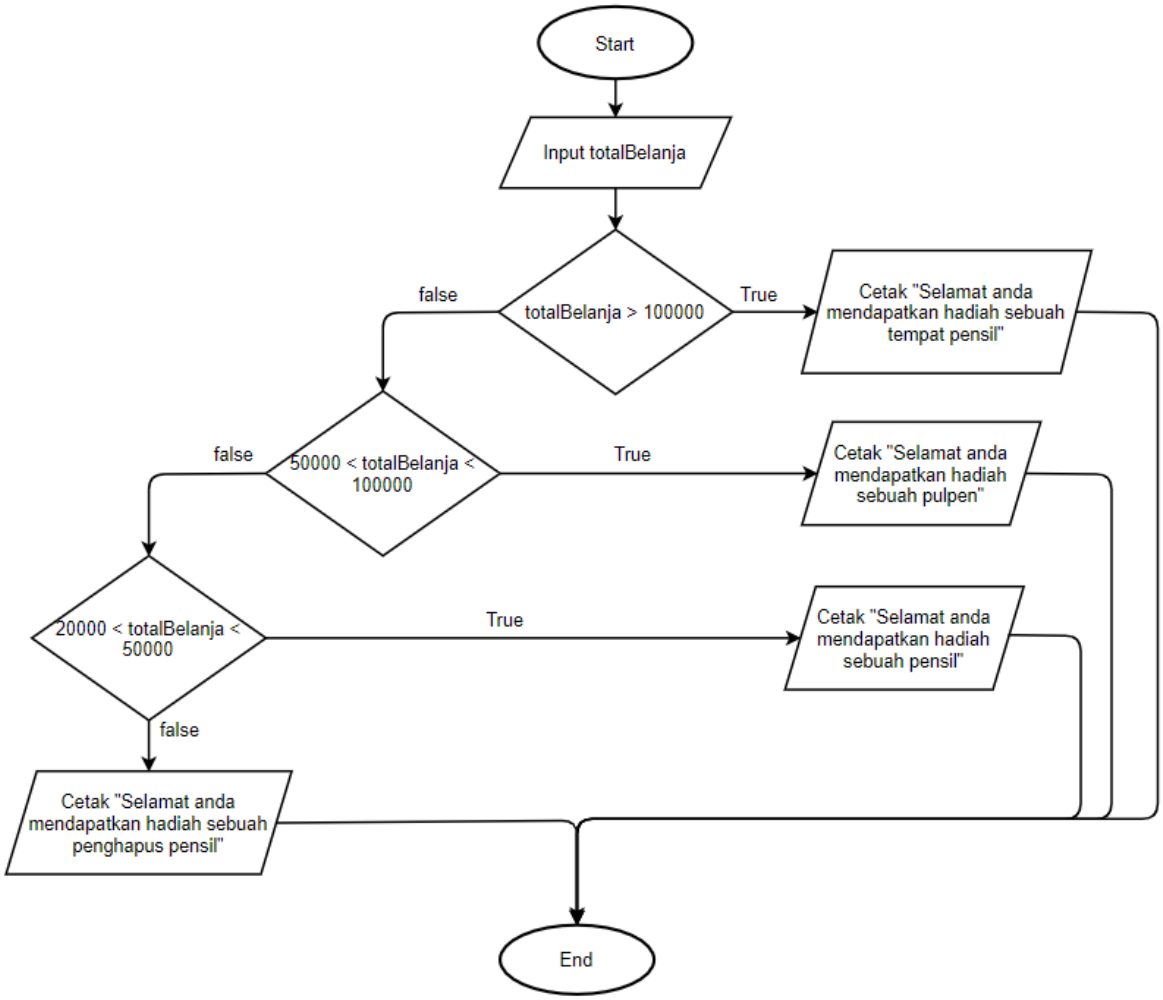
2.1.3 *Statement* `if-else if`

Pada penjelasan sebelumnya kita sudah menggabungkan *statement* `if-else` untuk mengatasi kondisi `true` dan `false`. Lalu bagaimana ketika kita memiliki kondisi yang lebih kompleks dan banyak. Maka kita bisa menggunakan *statement* `if-else if`. *Statement* `if-else if` digunakan ketika kita ingin membuat beberapa keputusan lain pada banyaknya kondisi yang akan diuji. *Statement* `else if` dapat kita tuliskan sebanyak yang kita inginkan sesuai dengan kondisi program yang akan dijalankan. Format penulisan *statement* `if-else if` adalah sebagai berikut.

```
if (ekspresiBoolean1)
{
    statement
    statement
    ...
}
else if (ekspresiBoolean2)
{
    statement
    statement
    ...
}
else if (ekspresiBoolean3)
{
    statement
    statement
    ...
}
else
{
    statement
    statement
    ...
}
```

Saat *statement* `if-else if` dieksekusi, pertama-tama program akan menguji `ekspresiBoolean1`. Jika `ekspresiBoolean1` terpenuhi dan bernilai `true`, maka program akan menjalankan *statement* didalam `if` dan mengabaikan kondisi-kondisi di bawahnya. Jika `ekspresiBoolean1` tidak terpenuhi dan bernilai `false`, maka program akan mengeksekusi *statement* kondisi selanjutnya yaitu `else if` dengan menguji variabel `ekspresiBoolean2`. Jika `ekpresiBoolean2` terpenuhi dan bernilai `true`, program akan mengeksekusi *statement* didalam `else if` dan mengabaikan kondisi-kondisi di bawahnya, begitu seterusnya.

Kita dapat mengambil contoh dengan mengembangkan program pada contoh sebelumnya. Toko buku akan memberikan sebuah hadiah berupa tempat pensil kepada pembeli jika nominal pembelian diatas $Rp 100.000$. Jika nominal pembelian diantara $Rp 50000 - Rp 100000$, toko buku akan memberikan hadiah berupa sebuah pulpen. Jika nominal pembelian diantara $Rp 20000 - Rp 50000$, toko buku akan memberikan hadiah berupa sebuah pensil. Selain itu, toko buku akan memberikan hadiah berupa penghapus pensil. *Flowchart* dari program tersebut adalah sebagai berikut.



Pada diagram *flowchart* diatas, akan dilakukan *input* nilai variabel `totalBelanja`. Selanjutnya akan dicek nilai dari variabel `totalBelanja` apakah lebih besar dari $Rp 100000$ atau tidak. Apabila nilai dari variabel `totalBelanja` lebih dari $Rp 100000$, maka program akan mencetak `Selamat anda mendapatkan hadiah sebuah tempat pensil`. Sedangkan jika nilai variabel `totalBelanja` diantara $Rp 50000 - Rp 100000$, maka program akan mencetak `Selamat anda mendapatkan sebuah pulpen`. Jika nilai variabel `totalBelanja` diantara $Rp 20000 - Rp 50000$, maka program akan mencetak `Selamat anda mendapatkan sebuah pensil`. Jika selain itu, maka program akan mencetak `Selamat anda mendapatkan hadiah sebuah penghapus pensil`. Berikut merupakan kode program dari pengecekan variabel `totalBelanja`.

Program (Hadiah.java)

```
import java.util.Scanner;

public class Hadiah
{
    public static void main(String[] args)
    {
        // Membuat variabel totalBelanja dan scanner
        int totalBelanja;
        Scanner scan = new Scanner(System.in);

        // Memasukkan nilai totalBelanja
        System.out.print("Total Belanja : Rp ");
        totalBelanja = scan.nextInt();

        if (totalBelanja > 100000)
        {
               System.out.println("Selamat anda mendapatkan hadiah sebuah tempat
pensil");
        }
        else if (totalBelanja > 50000 && totalBelanja < 100000)
        {
            System.out.println("Selamat anda mendapatkan hadiah sebuah pulpen");
        }
        else if (totalBelanja > 20000 && totalBelanja < 50000)
        {
            System.out.println("Selamat anda mendapatkan hadiah sebuah pensil");
        }
        else
        {
            System.out.println("Selamat anda mendapatkan hadiah " +
                               "sebuah penghapus pensil");
        }
    }
}
```

Output Program (Hadiah.java)

Jika *input* `totalBelanja = 120000`

```
Total Belanja : Rp 120000
Selamat anda mendapatkan hadiah sebuah tempat pensil
```

Jika *input* `totalBelanja = 70000`

```
Total Belanja : Rp 70000
Selamat anda mendapatkan hadiah sebuah pulpen
```

Jika *input* `totalBelanja = 30000`

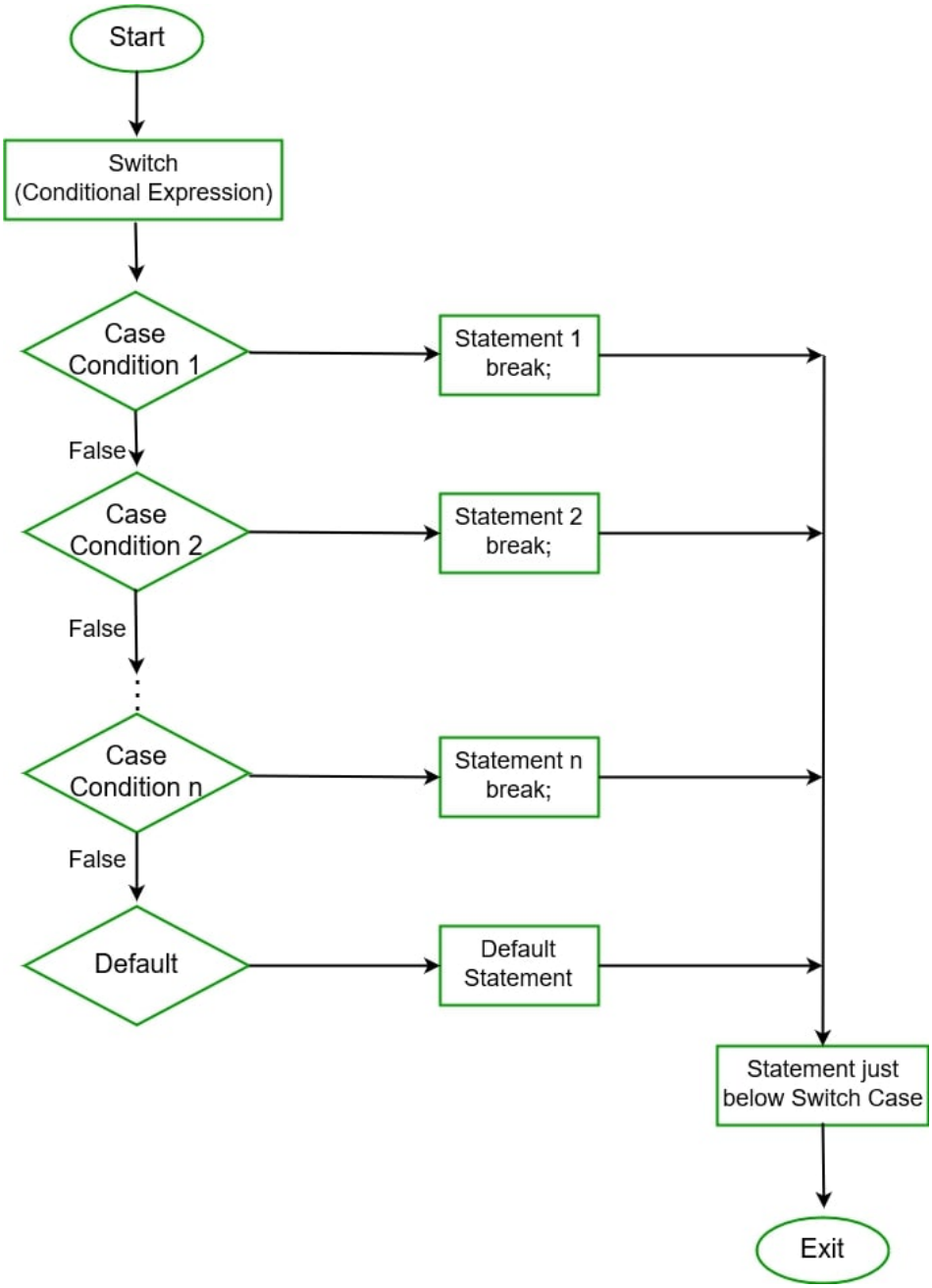
```
Total Belanja : Rp 30000
Selamat anda mendapatkan hadiah sebuah pensil
```

Jika *input* `totalBelanja = 15000`

```
Total Belanja : Rp 15000
Selamat anda mendapatkan hadiah sebuah penghapus pensil
```

2.1.4 *Statement* Switch

Saat kita ingin membuat sebuah struktur percabangan dari kondisi yang banyak, akan kurang efektif jika kita menggunakan percabangan `if`. Dalam kasus ini, kita bisa menggunakan *statement* `switch` untuk menggantikan percabangan `if`. Percabangan `switch` akan menggunakan *statement* `case` di dalam bloknya. *Statement* `case` digunakan untuk menguji nilai dari variabel yang akan diuji. *Statement* `case` dapat kita tuliskan sebanyak yang kita inginkan sesuai dengan kondisi program yang akan dijalankan. Jika dari banyaknya *statement* `case` di dalam percabangan `switch` tidak ada yang terpenuhi, maka *statement* `default` akan dijalankan. Berikut *flowchart* dari cara kerja *statement* `switch`.



Berikut merupakan bentuk umum penulisan *statement* `switch`.

```
switch (ekspresiUji)
{
    case nilai_1:
        statement
        statement
        ...
        break;

    case nilai_2:
        statement
        statement
        ...
        break;

    case nilai_n:
        statement
        statement
        ...
        break;

    /*
        case Default akan berjalan jika nilai dari ekspresiUji tidak
        sama dengan salah satu dari nilai case.
    */
    case Default:
        statement
        statement
        ...
        break;
}
```

Penggunaan percabangan `switch` diawali dengan menulis *keyword* `switch` dilanjutkan dengan `ekspresiUji` yang ditulis di dalam tanda kurung. Di dalam *statement* `switch`, tiap kondisi harus diawali dengan *keyword* `case` lalu dilanjutkan dengan kondisi yang diinginkan serta dalam setiap bagian `case` harus diakhiri dengan *keyword* `break`. *Keyword* `break` berfungsi untuk menghentikan blok dari sebuah kondisi `case`.

Saat *statement* `switch` dijalankan, nilai dari `ekspresiUji` akan dibandingkan dengan setiap `case` yang ada. Jika salah satu `case` mempunyai nilai yang sesuai dengan nilai dari `ekspresiUji`, maka blok *statement* `case` tersebut lah yang akan dijalankan. Sedangkan blok `default` akan dijalankan ketika tidak ada *statement* `case` yang dapat memenuhi nilai dari `ekspresiUji`.

Statement `switch` sendiri merupakan alternatif untuk *statement* `if-else if`. Struktur *statement* `switch` terlihat lebih rapih jika dibandingkan dengan *statement* `if-else if` dan kadang kala juga lebih efisien. Namun *statement* `switch` juga memiliki batasan, yaitu *statement* `switch` tidak bisa digunakan untuk kondisi yang lebih kompleks seperti perbandingan dengan tanda `<`, `>`, maupun dengan penggabungan kondisi. Beberapa kasus tersebut hanya bisa dieksekusi dengan menggunakan *statement* `if-else if`. Sehingga *statement* `switch` ini hanya cocok digunakan untuk operasi perbandingan sederhana, dimana nilai yang diperiksa hanya terdiri dari nilai yang tetap.

Pada contoh sebelumnya, kita menggunakan *statement* `if-else if` untuk menguji `totalBelanja`. Kasus tersebut tidak dapat kita eksekusi dengan menggunakan *statement* `switch` karena di dalamnya terdapat perbandingan menggunakan tanda `<` dan `>`. Berikut merupakan contoh untuk *statement* `switch` dengan menggunakan kasus penjumlahan dan perkalian suatu nilai.

Program (OperasiAngka.java)

```
public class OperasiAngka
{
    public static void main(String[] args)
    {
        int num = 10;

        switch(num * 5 + num)
        {
            case 20:
                System.out.println("Option 1: Nilai dari num adalah "+num);
                break;

            case 40:
                System.out.println("Option 2: Nilai dari num adalah "+num);
                break;

            case 60:
                System.out.println("Option 3: Nilai dari num adalah "+num);
                break;

            case 80:
                System.out.println("Option 4: Nilai dari num adalah "+num);

            default:
                System.out.println("Default Case: Nilai dari num adalah "+num);
                break;
        }
    }
}
```

Output Program (OperasiAngka.java)

```
Option 3: Nilai dari num adalah 60
```

Pada program di atas, nilai awal dari variabel `num` adalah `10`. Lalu dilakukan operasi matematika sehingga menghasilkan `num = num * 5 + num = 10 * 5 + 10 = 50 + 10 = 60`. Maka `case` yang dijalankan adalah `case` yang bernilai 60. Secara otomatis program akan mencetak *statement* yang ada di dalam `case 60:` dan mengakhiri percabangan `switch`.

`case` Tanpa *Statement* `break`

Statement `break` memegang peran penting dalam percabangan `switch`. Karena jika tidak menggunakan *statement* `break` dalam sebuah `case` maka program akan terus mengeksekusi *statement-statement* `case` seterusnya seperti contoh berikut.

Program (OperasiAngkaTanpaBreak.java)

```
public class OperasiAngkaTanpaBreak
{
    public static void main(String[] args)
    {
        int num = 10;

        switch(num * 5 + num)
        {
            case 20:
                System.out.println("Option 1: Nilai dari num adalah "+num);

            case 40:
                System.out.println("Option 2: Nilai dari num adalah "+num);

            case 60:
                System.out.println("Option 3: Nilai dari num adalah "+num);

            case 80:
                System.out.println("Option 4: Nilai dari num adalah "+num);

            default:
                System.out.println("Default Case: Nilai dari num adalah "+num);
        }
    }
}
```

Output Program (OperasiAngkaTanpaBreak.java)

```
Option 3: Nilai dari num adalah 60
Option 4: Nilai dari num adalah 60
Default Case: Nilai dari num adalah 60
```

Dapat dilihat bahwa *statement* `switch` berjalan begitu saja seperti tidak ada rem yang mengakibatkan program mengeksekusi semua *statement* `case` setelah *statement* `case` yang dijalankan. Seperti pada program di atas, program tersebut seharusnya mengeksekusi `case 60:` saja, namun karena setelah *statement* `case 60:` kita tidak memiliki *statement* `break;` maka program akan terus berjalan hingga *statement* `default:`.

2.1.5 Percabangan dalam percabangan (Nested)

Terkadang kita membutuhkan struktur percabangan yang lebih kompleks saat membuat program. Kita bisa saja memerlukan untuk menguji satu kondisi jika salah satu kondisi lainnya sudah terpenuhi. Pada kasus tersebut, kita bisa menggunakan percabangan dalam percabangan.

Sebagai contoh, kita akan menguji nilai mahasiswa dengan beberapa kondisi berikut.

- Jika `UTS > 80` dan `UAS > 80` mendapatkan nilai `A`.
- Jika `UTS > 80` dan `UAS < 80` mendapatkan nilai `A-`.
- Jika `UTS < 80` dan `UAS > 80` mendapatkan nilai `A-`.
- Jika `UTS < 80` dan `UAS < 80` mendapatkan nilai `B`.

Program (NestedIf.java)

```
import java.util.Scanner;

public class NestedIF
{
    public static void main(String[] args)
    {
        int uts, uas;
        Scanner scan = new Scanner(System.in);

        // Memasukkan nilai UTS
        System.out.print("UTS : ");
        uts = scan.nextInt();

        // Memasukkan nilai UAS
        System.out.print("UAS : ");
        uas = scan.nextInt();

        // Nested if untuk mengecek nilai UTS dan UAS.
        if(uts > 80)
        {
            if(uas > 80)
            {
                System.out.println("Anda Lulus Dengan Nilai A");
            }
            else
            {
                System.out.println("Anda Lulus Dengan Nilai A-");
            }
        }
        else
        {
            if(uas > 80)
            {
                System.out.println("Anda Lulus Dengan Nilai A-");
            }
            else
            {
                System.out.println("Anda Lulus Dengan Nilai B");
            }
        }
    }
}
```

Output Program (NestedIF.java)

Jika *input* `uts = 85` dan `uas = 95`

```
UTS : 85
UAS : 95
Anda Lulus Dengan Nilai A
```

Jika *input* `uts = 90` dan `uas = 75`

```
UTS : 90
UAS : 75
Anda Lulus Dengan Nilai A-
```

Jika *input* `uts = 75` dan `uas = 95`

```
UTS : 75
UAS : 95
Anda Lulus Dengan Nilai A-
```

Jika *input* `uts = 78` dan `uas = 75`

```
UTS : 78
UAS : 75
Anda Lulus Dengan Nilai B
```

Dalam contoh program di atas kita menggunakan percabangan dalam percabangan. Pertama kita akan memasukkan nilai dari variabel `uts` dan `uas`. Selanjutnya kita akan menguji variabel `uts`. Jika nilai variabel `uts > 80` maka akan bernilai `true` dan masuk ke percabangan yang selanjutnya yaitu percabangan yang akan menguji nilai dari variabel `uas`. Jika nilai dari variabel `uas > 80` maka akan bernilai `true` dan akan menampilkan *output* `Anda Lulus Dengan Nilai A`. Sedangkan jika bernilai `false` maka program akan menampilkan *output* `Anda Lulus Dengan Nilai A-`.

Lalu jika kondisi `uts > 80` bernilai `false` maka akan masuk ke percabangan `else`. Pada percabangan `else` terdapat percabangan lain yang akan menguji kondisi `uas > 80`. Jika bernilai `true` maka program akan menampilkan *output* `Anda Lulus Dengan Nilai A-`. Sedangkan jika kondisi `uas > 80` bernilai `false` maka program akan menampilkan *output* `Anda Lulus Dengan Nilai B`.

2.2 Struktur Perulangan

Perulangan dapat mengeksekusi kumpulan *code* di dalam bloknya asalkan kondisi untuk menjalankan perulangannya terpenuhi. Perulangan sangat membantu untuk mengeksekusi *statement* yang berulang. Dengan menggunakan perulangan, kita bisa menghemat waktu dalam penulisan program. Perulangan dalam pemrograman terbagi menjadi dua jenis, yaitu:

1. *Counted Loop* : Perulangan yang jumlah perulangannya terhitung atau tentu. Terdiri dari perulangan *for* dan *for each*.
2. *Uncounted Loop* : Perulangan yang jumlah perulangannya tidak terhitung atau tidak tentu. Terdiri dari perulangan *while do* dan *do while*.

2.2.1 *Counted Loop*

Perulangan `for`

Perulangan `for` digunakan ketika nilai dari variabel iterasinya sudah diketahui. Sama seperti jenis perulangan lainnya, perulangan `for` akan berjalan ketika kondisi perulangan nya bernilai `true` dan akan berhenti jika kondisi perulangannya bernilai `false`. Berikut merupakan bentuk umum dari perulangan `for`.

```
for (Inisialisasi; PengujiBatas; Pengupdate)
{
    statement;
    statement;
    ...
    ...
    ...
}
```

Perulangan `for` memiliki tiga elemen, yaitu:

1. **Inisialisasi** variabel *counter* yang menghitung banyaknya perulangan.
2. **Penguji Batas** yang akan menguji variabel *counter* terhadap suatu batas. Ketika variabel *counter* sudah mencapai batas, maka perulangan akan berhenti.
3. **Peng*update*** yang akan melakukan inkrementasi atau dekrementasi pada variabel *counter*.

Berikut adalah contoh perulangan `for`.

```
for (int counter = 1; counter <=5; counter++)
{
    System.out.println("Perulangan ke: " + counter);
}
```

Pada *statement* perulangan `for` diatas, `int counter = 1` merupakan *statement* inisialisasi, `counter <= 5` merupakan *statement* penguji batas lalu `counter++` merupakan *statement* peng*update*. Serta kita bisa memanggil variabel `counter` dan menggunakan nilainya di dalam *body* perulangan `for`. Seperti contoh diatas, kita menggunakan variabel `counter` untuk mencetak posisi perulangan dari `for`. Kita juga bisa menginisiasi variabel `counter` di luar perulangan.

```
int counter;

for (counter = 1; counter <=5; counter++)
{
    System.out.println("Perulangan ke: " + counter);
}
```

Karena variabel `counter` diinisiasi di luar perulangan, maka kita bisa memanggil nilai dari variabel `counter` di luar *body* perulangan. Berikut merupakan contoh pemanggilan variabel `counter` di luar *body* perulangan.

Program (PerulanganFor.java)

```
public class PerulanganFor
{
    public static void main(String[] args)
    {
        int counter;

        for (counter = 1; counter <=5; counter++)
        {
            System.out.println("Nilai variabel counter: " + counter);
        }

        System.out.println(counter);
    }
}
```

Output Program (PerulanganFor.java)

```
Nilai variabel counter: 1
Nilai variabel counter: 2
Nilai variabel counter: 3
Nilai variabel counter: 4
Nilai variabel counter: 5
6
```

Perhatikan hasil program di atas, *statement* `System.out.println(counter);` akan mencetak nilai variabel `counter` ketika perulangan sudah selesai yang mana nilainya adalah `6`. Perulangan `for` berjalan ketika nilai variabel `counter<=5` yang mana berarti perulangan akan berhenti ketika nilai variabel `counter>5`. Ketika nilai variabel `counter` mencapai nilai `counter=5`, maka perulangan masih berjalan. Namun ketika nilai variabel `counter=6` maka perulangan otomatis akan langsung berhenti karena sudah tidak memenuhi *statement* penguji batas, yaitu `counter<=5`. Maka nilai variabel `counter` terakhir ketika perulangan terhenti adalah `counter=6`, oleh karena itu ketika variabel `counter` dipanggil di luar *body* perulangan nilainya akan menghasilkan `counter=6`.

Bentuk Lain Pengupdate

Peng*update*an dalam perulangan `for` tidak hanya terbatas menggunakan inkerementasi atau dekerementasi saja, tetapi kita bisa menentukan variabel peng*update* sebesar yang kita inginkan. Seperti contoh berikut.

Program (PerulanganFor.java)

```
public class PerulanganFor
{
    public static void main(String[] args)
    {
        int num;

        for (num = 10; num >= 0; num -= 2)
        {
            System.out.println(num);
        }
    }
}
```

Output Program (PerulanganFor.java)

```
10
8
6
4
2
0
```

Perulangan *for each*

Perulangan *for each* digunakan khusus untuk mengiterasi elemen-elemen yang ada di dalam sebuah *array*. *Array* merupakan variabel yang dapat menyimpan lebih dari satu nilai dan memiliki indeks. Perulangan *for each* pada Java dilakukan juga dengan *keyword* `for`. Berikut merupakan bentuk dasar dari perulangan *for each*.

```
for (tipeData namaVariabel: namaArray)
{
    statement;
    statement;
    ...
    ...
    ...
}
```

Contoh penulisannya adalah sebagai berikut.

```
for ( int elemen : dataArray )
{
    // statement yang akan diulang
}
```

Variabel `elemen` akan menyimpan nilai dari *array*. Perulangan *for each* di atas dapat dibaca "Untuk setiap `elemen` dalam `dataArray`, maka akan dilakukan *statement* sesuai yang dituliskan di dalam blok kode". Berikut contoh perulangan *for each*.

Program (PerulanganForeach.java)

```
public class PerulanganForeach
{
    public static void main(String[] args)
    {
        // Membuat array
        int himpunanAngka[] = {3,1,10,19,15};

        /*
         Menggunakan perulangan For each untuk menampilkan nilai
         setiap elemen dari array himpunanAngka
        */
        for( int item : himpunanAngka )
        {
            System.out.print(item + " ");
        }
    }
}
```

Output Program (PerulanganForeach.java)

```
3 1 10 19 15
```

2.2.2 *Uncounted Loop*

Perulangan *while do*

Perulangan *while do* merupakan salah satu dari beberapa perulangan di Java. Perulangan *while do* menggunakan *keyword* `while` dalam penulisannya. Perulangan *while do* akan bekerja saat kondisi dari `while` bernilai `true` dan akan berhenti jika nilai dari `while` berubah menjadi `false`. Berikut merupakan bentuk umum dari perulangan *while do*.

```
while (ekspresiBoolean)
{
    statement
    statement
    ...
    ...
    ...
}
```

Logika dari perulangan *while do* adalah perintah perulangan akan berjalan ketika `ekspresiBoolean` bernilai `true` dan perulangan akan berhenti ketika `ekspresiBoolean` bernilai `false`. Cara kerja perulangan *while do* sama seperti percabangan `if` yang dieksekusi berulang kali. Berikut merupakan contoh program dengan perulangan *while do*.

Program (PerulanganWhile.java)

```
import java.util.Scanner;

public class PerulanganWhile
{
    public static void main(String[] args)
    {
        // Membuat variabel dan scanner
        boolean running = true;
        int counter = 1;
        String jawab;
        Scanner scan = new Scanner(System.in);

        // Melakukan perulangan
        while( running )
        {
            System.out.println("Perulangan ke - "+counter);
            System.out.println("=======================");
            System.out.println("Apakah anda ingin keluar?");
            System.out.print("Jawaban [ya/tidak]> ");

            jawab = scan.nextLine();

            /*
             Pengecekan jawaban, kalau jawab=ya maka running akan
             bernilai false dan perulangan akan berhenti.
            */
```

```
            if( jawab.equalsIgnoreCase("ya") )
            {
                running = false;
            }
            System.out.println();
            counter++;
        }
    }
}
```

Output Program (PerulanganWhile.java)

Jika variabel `jawab` langsung bernilai `ya`.

```
Perulangan ke - 1
=======================
Apakah anda ingin keluar?
Jawaban [ya/tidak]> ya
```

Jika variabel `jawab` bernilai `tidak` sebanyak dua kali lalu nilai variabel `jawab` diganti menjadi `ya`.

```
Perulangan ke - 1
=======================
Apakah anda ingin keluar?
Jawaban [ya/tidak]> tidak

Perulangan ke - 2
=======================
Apakah anda ingin keluar?
Jawaban [ya/tidak]> tidak

Perulangan ke - 3
=======================
Apakah anda ingin keluar?
Jawaban [ya/tidak]> ya
```

Perhatikan *output* dari program di atas. Program tersebut dapat berjalan sebanyak yang pengguna inginkan. Pengguna dapat melakukan perulangan tersebut sebanyak 1 kali atau 5 kali atau bahkan 100 kali. Selama nilai dari variabel `jawab=tidak` maka program akan terus melakukan perulangan.

Perulangan *while do* juga dapat melakukan *counted loop* selama kondisi yang dimasukkan bernilai tentu. Berikut perulangan *while do* jika kondisi yang dimasukkan bernilai tentu.

Program (LoopWhile.java)

```
/*
    Program ini menggunakan loop while untuk
    mencetak jumlah perulangan yang dijalani sebanyak lima kali.
*/
public class Loopwhile
{
    public static void main(String[] args)
    {
        int posisiPerulangan = 1;
```

```
        /*
        jika posisiPerulangan bernilai true,
        maka eksekusi statement-statemnt di bawah ini
        */
        while (posisiPerulangan <= 5)
        {
            System.out.println("Perulangan ke: " + posisiPerulangan);
            posisiPerulangan++; // iterasi nilai posisiPerulangan
        }

        System.out.println("Selesai!");
    }
}
```

Output Program (LoopWhile.java)

```
Perulangan ke: 1
Perulangan ke: 2
Perulangan ke: 3
Perulangan ke: 4
Perulangan ke: 5
Selesai!
```

Program diatas menggunakan perulangan *while do* yang dijalankan sebanyak lima kali. Setiap pengulangan dari perulangan disebut sebagai `iterasi`. Variabel `posisiPerulangan` diinisiasi dengan nilai `1` dan nilainya akan diinkrementasi setiap perulangan berjalan. Maka pada iterasi ke `5`, nilai `posisiPerulangan` menjadi 6 dan kondisi perulangan *while do* yaitu `(posisiPerulangan <= 5)` akan menjadi `false` dan perulangan pun akan berhenti.

Tabel berikut menunjukan perubahan variabel `posisiPerulangan` setiap proses perulangan berjalan.

Iterasi	nilai variabel `posisiPerulangan`	Pengecekan `posisiPerulangan <= 5`	Eksekusi
1	1	`true`	- Cetak `Perulangan ke: 1` - `posisiPerulangan++` menghasilkan `posisiPerulangan = 2`
2	2	`true`	- Cetak `Perulangan ke: 2` - `posisiPerulangan++` menghasilkan `posisiPerulangan = 3`
3	3	`true`	- Cetak `Perulangan ke: 3` - `posisiPerulangan++` menghasilkan `posisiPerulangan = 4`
4	4	`true`	- Cetak `Perulangan ke: 4` - `posisiPerulangan++` menghasilkan `posisiPerulangan = 5`
5	5	`true`	- Cetak `Perulangan ke: 5` - `posisiPerulangan++` menghasilkan `posisiPerulangan = 6`
6	6	`false`	**LOOP BERHENTI**

Pada program di atas terdapat *statement* inkrementasi `posisiPerulangan++`. *Statement* ini sangatlah penting karena bertujuan untuk menambah jumlah nilai variabel `posisiPerulangan` yang akan membuat perulangan akan berhenti saat kondisi `while` sudah bernilai `false`.

Jika kita tidak melakukan inkrementasi pada variabel `posisiPerulangan`, maka perulangan akan terus berjalan dan tidak pernah berhenti yang mengakibatkan program kita mengalami *Infinite Loop*. Sesuai dengan namanya, *Infinite Loop* merupakan suatu perulangan yang tidak akan pernah berhenti. Hal tersebut dikarenakan kondisi dari perulangan akan selalu bernilai `true` dan tidak akan pernah bernilai `false`.

Pada program di atas kita melakukan inkrementasi pada variabel `posisiPerulangan` agar nilainya selalu bertambah dan akan membuat kondisi perulangan `posisiPerulangan <= 5` menjadi `false` dan membuat proses perulangan terhenti. Jika *statement* `posisiPerulangan++` tidak ditulis, maka variabel `posisiPerulangan` akan selalu bernilai 1 dan program akan mengalami *Infinite Loop*.

Perulangan *do-while*

Perulangan *do-while* mirip dengan perulangan *while do*. Pembedanya adalah perulangan *do-while* akan mengeksekusi satu kali perulangan terlebih dahulu baru akan menguji kondisi perulangannya. Sedangkan perulangan *while do* menguji kondisi perulangannya terlebih dahulu, jika kondisi bernilai `true` maka perulangan akan dieksekusi. Berikut merupakan bentuk umum dari *statement do-while*.

```
do
{
    statement
    statement
    ...
    ...
    ...
}
while(ekspresiBoolean);
```

Berikut merupakan contoh perbedaan antara perulangan *while do* dan *do-while*.

Program (WhileDo.java)

```
public class WhileDo
{
    public static void main(String[] args)
    {
        int num = 1;
        while (num < 0)
        {
            System.out.println(num);
        }
    }
}
```

Dengan menggunakan *statement* perulangan *while do* seperti diatas, maka *statement* `println` tidak akan tereksekusi karena nilai dari ekspresi perulangan *while do* adalah *false*. Oleh karena itu, program tersebut tidak akan menghasilkan *output* apapun.

Sebaliknya jika kita menggunakan *statement* perulangan *do-while*, maka *statement* `println` akan tereksekusi karena pada perulangan *do-while* pengecekkan nilai ekspresi perulangan dilakukan setelah *statement* di dalam *body* perulangan dijalankan. Seperti pada contoh berikut.

Program (DoWhile.java)

```
public class WhileDo
{
    public static void main(String[] args)
    {
        int num = 1;
        do{
            System.out.println("Halo");
        }
        while (num < 0);
    }
}
```

Output Program (DoWhile.java)

```
Halo
```

2.2.3 *Nested Loop* (Perulangan Bersarang)

Perulangan bersarang atau biasa disebut *Nested Loop* merupakan proses dimana terdapat perulangan di dalam perulangan lain. *Nested Loop* berguna untuk melakukan operasi perulangan yang di dalamnya masih perlu dilakukan operasi perulangan lagi.

Sebagai contoh, kita ingin membuat program yang mencetak *pattern* berformat persegi 3x3 dengan angka 1,2 dan 3. Kita bisa memanfaatkan perulangan `for` bersarang seperti berikut.

Program (PatternPersegi.java)

```
public class PatternPersegi
{
    public static void main(String args[])
    {
        // Inisialisasi jumlah baris dan kolom
        final int maxBaris = 3;
        final int maxKolom = 3;

        // Melakukan perulangan untuk setiap baris
        for(int i = 1;i <= maxBaris; i++)
        {
            // Melakukan perulangan untuk setiap kolom
            for(int j = 1; j <= maxKolom; j++)
            {
                // Mencetak nilai indeks kolom
                System.out.print(j + " ");
            }

            // Mencetak perpindahan baris baru
            System.out.println("");
        }
    }
}
```

Output Program (PatternPersegi.java)

```
1 2 3
1 2 3
1 2 3
```

Dari program diatas diketahui jumlah maksimal dari variabel `maxBaris` dan `maxKolom` adalah sama, yaitu tiga. Karena memiliki jumlah baris dan kolom yang sama, maka *output* program kita bisa berbentuk persegi.

Perulangan bersarang sering juga digunakan pada *array* multi dimensi. Jenis perulangan di dalam perulangan bisa berbeda, misalnya pada *body* perulangan *while do* terdapat perulangan *for*, dan sebagainya.

2.3 *Method*

2.3.1 *Method*

Method merupakan suatu blok yang berisi kumpulan *statement* yang diberikan nama. *Method* bisa dieksekusi dengan memanggil nama *method*. *Method* digunakan untuk melakukan tugas-tugas tertentu dan biasanya *method* dipanggil sebagai *function*. Sejauh ini kita sudah mengetahui dua buah *method*, yaitu.

1. *Method* bernama `main` yang ditulis pada *statement* `public static void main`.
2. *Method* `System.out.println()` yang digunakan untuk menampilkan *output*.

Berikut merupakan bentuk umum dari *method*.

```
accessSpecifier returnType namaMethod (argumen)
{
    statement
    statement
    ...
    ...
    ...
}
```

Berikut merupakan contoh *method* sederhana.

```
// Header method
public static void haloDunia()
{
    // Body method
    System.out.println("Halo Dunia!");
}
```

Berikut penjelasan dari komponen-komponen kode di atas.

`public` merupakan *access specifier* dari *method*. Dengan menggunakan `public`, *method* kita bisa diakses oleh seluruh *class*.

`static` merupakan *access modifier* dari *method*. Dengan menggunakan `static`, *method* akan menjadi bagian dari *main class*. *Access specifier* akan kita bahas lebih lengkap pada subbab lainnya.

`void` merupakan tipe data yang akan dikembalikan oleh *method*. Tipe data `void` tidak mengembalikan nilai apapun. Tipe data akan kita bahas pada subbab lainnya.

`haloDunia()` merupakan nama *method*.

Setelah *method* berhasil dibuat, selanjutnya adalah pemanggilan *method* `haloDunia()` di dalam *method* `main`. Berikut cara pemanggilannya.

Program (TestHaloDunia.java)

```
public class TestHaloDunia
{
    public static void main(String[] args)
    {
        // Memanggil method haloDunia()
```

```
        haloDunia();
    }

    // Membuat method haloDunia
    public static void haloDunia()
    {
        System.out.println("Halo Dunia!");
    }
}
```

Output Progam (TestHaloDunia.java)

```
Halo Dunia!
```

2.3.2 *Argument* ke *Method*

Method dapat menerima nilai-nilai untuk diproses. Nilai-nilai yang diberikan ke *method* ini disebut sebagai *argument*. *Argument* tersebut dapat menerima tipe data primitif maupun non primitif. Sebuah *method* dapat menerima *argument* sebanyak apapun yang diperlukan. Tiap variabel yang dibuat pada kolom *argument* harus dipisahkan dengan simbol `,` (koma). Berikut adalah contoh penulisan definisi *method* yang menerima satu *argument* non primitif `String`.

```
public static void sapaNama(String nama)
{
    System.out.println("Halo " + nama);
}
```

Perhatikan pada *header* definisi *method* di atas, di dalam tanda kurung kita menuliskan deklarasi variabel berikut: `String nama`. Pada bagian ini, kita mendeskripsikan sebuah variabel yaitu `nama` dengan tipe data `String`. *Argument* sendiri diibaratkan sebagai tempat menampung nilai yang akan di*input* ke *method*. *Statement* berikut adalah pemanggilan *method* `sapaNama` dengan memberikan nilai `"Juki"` sebagai *argument*.

Program (PemanggilanMethod.java)

```
public class PemanggilanMethod
{
    public static void main(String[] args)
    {
        // Memanggil method sapaNama() dengan memberikan argument ke dalamnya
        sapaNama("Juki");
    }

    // Membuat method sapaNama()
    public static void sapaNama(String nama)
    {
        System.out.println("Halo " + nama);
    }
}
```

Output Progam (PemanggilanMethod.java)

```
Halo Juki
```

Berikut adalah contoh penulisan definisi *method* yang menerima dua *argument* primitif `int`.

Program (MethodDuaArgument.java)

```
public class MethodDuaArgument
{
    public static void main(String[] args)
    {
        // Memanggil method hitungUmur() dengan memberikan argument ke dalamnya
        hitungUmur(2023, 2002);
    }

    // Membuat method hitungUmur()
    public static void hitungUmur(int tahunSekarang, int tahunLahir)
    {
        int umur = tahunSekarang - tahunLahir;
        System.out.println("Umur kamu adalah: " + umur);
    }
}
```

Output Progam (MethodDuaArgument.java)

```
Umur kamu adalah: 21
```

Pada program di atas, didefinisikan *method* `hitungUmur` dengan dua *argument* yang bertipe data `int` yaitu `tahunSekarang` dan `tahunLahir`. Jika kedua *argument* tersebut diberikan nilai `2023` pada variabel `tahunSekarang` dan `2002` pada variabel `tahunLahir`, maka ketika *method* `main()` memanggil *method* `hitungUmur()` akan menghasilkan *output* `Umur kamu adalah: 21`.

2.3.3 Nilai *Return* pada *Method*

Pada subbab sebelumnya kita sudah membuat *method* sederhana yang hanya menampilkan *output* `Halo + nama` ketika dieksekusi. Dikarenakan *method* sebelumnya tidak mengembalikan nilai apapun, maka tipe data *return* yang digunakan untuk *method* tersebut adalah tipe data `void`. Jika *method* yang dibuat akan mengembalikan suatu nilai, maka tipe data `void` bisa diganti menjadi tipe data lain. Seperti tipe data primitif yaitu `int`, `char` dan tipe data non-primitif yaitu `object`, `String`. Berikut merupakan contoh *method* yang akan mengembalikan tipe data primitif `int` saat dieksekusi.

```
public static int penjumlahanBilangan(int angka1, int angka2)
{
   return angka1 + angka2;
}
```

Method `penjumlahanBilangan()` memiliki tipe data *return* `int` dan memiliki dua buah *argument* yaitu `angka1` dan `angka2`. *Method* `penjumlahanBilangan()` akan mengembalikan nilai dari hasil penjumlahan `angka1` dan `angka2` saat dipanggil. Selanjutnya kita bisa memanggil *method* tersebut dalam *method* `main`.

Program (JumlahBilangan.java)

```
public class JumlahBilangan
{
    public static void main(String[] args)
```

```
    {
        int num1 = 7;
        int num2 = 10;
        /*
                  Variabel hasil akan menyimpan nilai return dari method
penjumlahanBilangan()
        */
        int hasil = penjumlahanBilangan(num1, num2);

        System.out.println(num1 + " ditambah " + num2 + " = " + hasil);
    }

    // Membuat method penjumlahanBilangan()
    public static int penjumlahanBilangan(int angka1, int angka2)
    {
        return angka1 + angka2;
    }
}
```

Output Program (JumlahBilangan.java)

```
7 ditambah 10 = 17
```

2.3.4 Lingkup Variabel

Lingkup Variabel atau *Variable Scope* merupakan jangkauan variabel pada program dapat diakses. Variabel hanya dapat diakses di tempat dimana ia dideklarasikan. Tempat variabel tersebut di deklarasikan disebut sebagai *scope* dari variabel tersebut. Misalkan terdapat variabel `diameter` dalam *method* `getDiameter()`, lalu variabel `diameter` tersebut ingin diakses di luar *method* `getDiameter()`. Maka akan didapatkan *error*. Perhatikan potongan program berikut.

```
public static int getDiameter(int diameter)
{
    return diameter;
}

public static int cetakDiameter()
{
    return diameter; //Error
}
```

Variabel Lokal

Variabel lokal merupakan variabel yang dideklarasikan dalam sebuah *method*, perulangan ataupun percabangan. Lingkup dari variabel lokal hanya sebatas di dalam tempat dimana variabel tersebut dideklarasikan. Yang berarti variabel lokal hanya dapat diakses di dalam bagian *body method*, perulangan ataupun percabangan tempat variabel tersebut dideklarasikan dan tidak dapat diakses oleh *statement* lain di luar tempat mereka dideklarasikan.

Variabel Global

Variabel global merupakan variabel yang dapat diakses oleh seluruh bagian dari program. Variabel global umumnya dideklarasikan di atas *method* `main`. Tujuan dideklarasikan di atas *method* `main` adalah agar variabel tersebut dapat diakses oleh seluruh bagian program. Berikut merupakan contoh program menggunakan variabel global pada java.

Program (TestGlobal.java)

```
public class TestGlobal
{
    // Deklarasi dan inisiasi variabel global
    public static String nama = "Agung";

    public static void main(String[] args)
    {
        // Mengakses variabel nama di dalam method main
        System.out.println("Hai " + nama);

        // Memanggil method sebutNama()
        sebutNama();
    }

    // Membuat method sebutNama()
    public static void sebutNama()
    {
        // Mengakses variabel nama di dalam method sebutNama()
        System.out.println("Halo " + nama);
    }
}
```

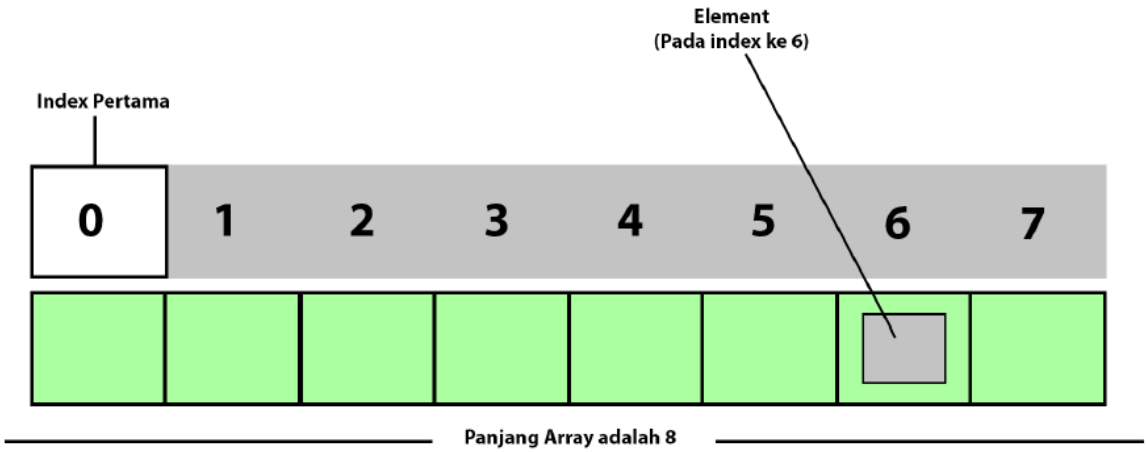
Output (TestGlobal.java)

```
Hai Agung
Halo Agung
```

Perlu diperhatikan bahwa pendeklarasian variabel global harus di dalam *class* dari program. Gunakan *keyowrd* `public` jika ingin variabel tetap dapat diakses oleh *class* lain yang berada dalam satu *package*. Gunakan *keyword* `private` jika ingin variabel hanya dapat diakses didalam *class*nya.

2.4 *Array*

Dalam bahasa pemrograman, umumnya *array* adalah sebuah koleksi data yang memiliki tipe data sejenis yang memiliki lokasi memori yang berdekatan. *Array* dalam Java adalah sebuah *object* yang berisi data-data yang memiliki tipe data sejenis. Elemen-elemen dari *array* disimpan di lokasi memori yang berdekatan. Pada Java, *array* berbasis *index* dimulai dari angka nol. Berikut merupakan gambaran dari *array* dalam Java.



Berikut merupakan bentuk umum dari *array*:

```
// Untuk deklarasi dan instansiasi array sekaligus
tipeData[] namaObject = new tipeData[panjangArray];

// Untuk deklarasi array
tipeData[] namaObject;

// Untuk instansiasi array
namaObject = new tipeData[panjangArray];
```

Untuk membuat *array*, pertama yang harus dilakukan adalah mendeklarasikan array. *Statement* `tipeData[]` merupakan tipe data dari *object array*. Tipe data tersebut dapat berupa tipe data primitif dan non primitif. *Statement* `namaObject` merupakan nama yang akan diberikan untuk *object array*. *Array* dapat langsung diinstansiasi saat proses pendeklarasian atau bisa juga dipisahkan antara instansiasi dan deklarasinya. Proses instansiasi *array* menggunakan *keyword* `new` yang berfungsi untuk membuat Java *object* dan mengalokasikan memori untuk *object array*.

Berikut merupakan contoh membuat *array* bertipe data *String* dengan panjang *array* `7`.

```
String[] namaHari = new String[7];
```

Menginisialisasi Array

Setelah berhasil mendeklarasikan dan menginstansiasi *array*, selanjutnya adalah memasukkan nilai-nilai ke dalam *array* yang telah dibuat. Memasukan nilai ke *array* dapat dilakukan dengan satu persatu atau sekaligus. Berikut merupakan contoh menginisialisasi *array* dengan cara satu per satu.

```
// Menginisialisasi array satu per satu
namaHari[0] =  "Senin";
namaHari[1] = "Selasa";
namaHari[2] = "Rabu";
```

Menginisialisasi *array* secara satu per satu dapat dilakukan dengan mengkases *array* yang telah dibuat dengan memanggil *array* tersebut. Pada kasus ini, nama *array* adalah `namaHari` lalu selanjutnya tentukan *index* berapa yang akan diakses dan diberi nilai.

```
// Mengakses array dan menginisialisasi nilai pada array index ke-0
namaHari[0] = "Senin";
```

Statement `namaHari[0] = "Senin";` berfungsi untuk mengakses *index* ke `0` pada *array* `namaHari` dan menginisialisasikan nilainya menjadi `"Senin"`.

Menginisialisasi *array* secara sekaligus hanya dapat dilakukan dengan menginisialisasi pada saat proses deklarasi. Berikut merupakan potongan kode untuk menginisialisasi *array* secara sekaligus.

```
String[] namaHari = {"Senin", "Selasa", "Rabu", "Kamis", "Jumat", "Sabtu",
"Minggu"};
```

Jika *array* sudah diinisalisasi, maka program dapat mencetak nilai dari *array* pada *index* tertentu dengan menggunakan *statement* `println()`.

```
// Mencetak nilai dari array namaHari pada index ke-0
System.out.println(namaHari[0]);
```

Program (ArrayNilai.java)

```
import java.util.Scanner;

public class ArrayNilai
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);

        System.out.println("Program Hitung Rata-Rata Nilai");
        System.out.println("==============================");
        System.out.print("Masukkan jumlah nilai yang ingin dihitung: ");

        // Masukkan nilai variabel jumlahNilai sebagai ukuran panjang array
        int jumlahNilai = scan.nextInt();

        // Deklarasi dan instansiasi arrayNilai
        int[] arrayNilai = new int[jumlahNilai];

        /*
         Perulangan for untuk memasukkan elemen - elemen ke dalam
         array secara satu per satu
        */
        for(int i = 0; i < jumlahNilai; i++)
        {
            System.out.print("Masukkan nilai: ");
            arrayNilai[i] = scan.nextInt();
        }

        System.out.print("List Nilai: ");

        // Perulangan forEach untuk mencetak setiap elemen dari arrayNilai
        for (int nilai : arrayNilai)
        {
            System.out.print(" " + nilai);
        }
    }
}
```

Output Program (ArrayNilai.java)

```
Program Hitung Rata-Rata Nilai
==============================
Masukkan jumlah nilai yang ingin dihitung: 3
Masukkan nilai: 70
Masukkan nilai: 80
Masukkan nilai: 65
List Nilai:  70 80 65
```

Pada program diatas, *array* `arrayNilai[]` digunakan sebagai tempat untuk menampung nilai-nilai yang akan di*input*kan oleh *user*. Selanjutnya digunakan perulangan `for` untuk memasukkan nilai setiap elemen ke dalam *array* secara satu per satu. Di dalam *body* perulangan `for` terdapat *statement* `arrayNilai[i] = scan.nextInt();` yang berfungsi untuk meng*input* nilai setiap elemen *array*.

Setelah semua nilai setiap elemen dimasukkan, selanjutnya akan dicetak nilai dari setiap elemen `arrayNilai[]` dengan menggunakan perulangan `forEach`. Di dalam *body* perulangan `forEach` terdapat *statement* `System.out.print(" " + nilai);` yang berfungsi untuk mencetak nilai setiap elemen *array*.

2.4.1 Menggunakan *Array* pada *Method*

Selain tipe data primitif, kita juga dapat memberi tipe data non primitif pada *argument method* maupun tipe data nilai *return* dari *method*. *Array* merupakan salah satu tipe data non primitif yang bisa menjadi *argument* maupun tipe data nilai *return* dari *method*.

Array* sebagai *Argument Method

Sebelumnya tanpa disadari, kita sudah membuat sebuah *method* yang memiliki *array* bertipe data *String* pada *argument*nya. *Method* ini adalah *method* `main`.

```
public static void main(String[] args)
{
    statement
    statement
    ...
    ...
    ...
}
```

Method `main` mempunyai satu *argument* yang bertipe data *array String* dan memiliki nama variabel `args`. Kita bisa mengakses dan memasukkan nilai bertipe data `String` pada variabel `args`. Selain itu kita juga bisa membuat *method* sendiri yang memiliki tipe data *array* sebagai *argument*nya. Berikut merupakan program membuat *method* dengan *array* sebagai *argument*nya.

Program (DemoArray.java)

```
public class DemoArray
{
    public static void main(String[] args)
    {
        // Membuat array yang menampung jenis-jenis buah pisang
            String[] buahPisang = {"Pisang Ambon", "Pisang Sunpride", "Pisang
Tanduk"};
```

```
        // Memasukkan array buahPisang sebagai argument method namaBuah
        namaBuah(buahPisang);
    }

    // Method nama buah memiliki array bertipe data String pada argumentnya
    public static void namaBuah(String[] arrayBuah)
    {
        for (String buah : arrayBuah)
        {
            System.out.println(buah);
        }
    }
}
```

Output Program (DemoArray.java)

```
Pisang Ambon
Pisang Sunpride
Pisang Tanduk
```

Nilai *return Method* bertipe data *Array*

Pengembalian *array* pada *method* harus menggunakan *array* yang digunakan untuk deklarasi *method*. Berikut merupakan contoh program pengembalian *array* pada *method*.

```
public static int[] arrayBilangan(int batasBilangan)
{
    // Membuat variabel array
    int[] bilanganArray = new int[batasBilangan + 1];

    // Memasukkan nilai untuk setiap elemen array
    for (int i = 0; i <= batasBilangan; i++)
    {
        // Memasukkan nilai ke variabel bilanganArray
        bilanganArray[i] =  i;
    }

    // Mengembalikan nilai bilanganArray untuk method arrayBilangan()
    return bilanganArray;
}
```

Pada potongan kode program di atas, kita bisa melihat bagaimana cara membuat sebuah *method* yang mengembalikan nilai bertipe data *array*. Pada bagian kode `public static int[] arrayBilangan(int batasBilangan)`, *statement* `int[]` menyatakan *method* `arrayBilangan` akan mengembalikan nilai *array* yang bertipe data `int`.

2.4.2 *Array String*

Pada subbab sebelumnya kita telah membuat *array* bertipe data *String*. Pada subbab ini kita akan bahas lebih *detail* mengenai *array String*. Berikut merupakan contoh *array* bertipe data *String* pada contoh subbab sebelumnya.

```
String [] namaHari = new String[7];
namaHari[0] = "Senin";
namaHari[1] = "Selasa";
namaHari[2] = "Rabu";
namaHari[3] = "Kamis";
namaHari[4] = "Jumat";
namaHari[5] = "Sabtu";
namaHari[6] = "Minggu";
```

Perlu diingat bahwa setiap elemen dalam *array String* merupakan sebuah referensi ke *object String*. *Array String* akan menyimpan referensi dari *object String* ke memori. Hal ini disebabkan karena tipe data *String* merupakan sebuah *object*. Elemen `namaHari[0]` menyimpan sebuah referensi dari *object String* yang bernilai `Senin`. Elemen `namaHari[1]` menyimpan sebuah referensi dari *object String* yang bernilai `Selasa`. Elemen `namaHari[2]` menyimpan sebuah referensi dari *object String* yang bernilai `Rabu` dan seterusnya.

Elemen *array* bertipe data *String* yang belum diinisialisasi otomatis akan memiliki nilai `null`. Nilai `null` disebabkan karena elemen tersebut tidak menampung referensi ke *object* apapun. Berikut merupakan cotoh penulisan kode yang menghasilkan elemen *array* bertipe data *String* bernilai `null`.

```
// Mendeklarasi sebuah array yang bernama arrayNull.
String[] arrayNull = new String [3];

// Mencetak nilai elemen arrayNull index ke-0.
System.out.println(arrayNull[0]);
```

Perhatikan contoh potongan kode program di atas. Pada kode tersebut telah dideklarasikan sebuah *array* bertipe data *String* yang diberi nama sebagai `arrayNull`. Selanjutnya yang dilakukan oleh program adalah mencetak nilai elemen dengan *index* ke-0 dari *array* `arrayNull`. Pada proses ini, program tersebut akan menghasilkan *output* `null`. Hal ini dikarenakan pada program di atas hanya dilakukan deklarasi *array* tanpa dilakukan inisialisasi nilai setiap elemen *array*. Berikut merupakan contoh program menggunakan *array String*.

Program (DemoArrayString.java)

```
import java.util.Scanner;

public class DemoArrayString
{
    public static void main(String[] args)
    {
        // Membuat Scanner baru
        Scanner scan = new Scanner(System.in);

        // Meminta user menentukan jumlah murid untuk ukuran array
        System.out.print("Masukkan jumlah murid: ");
        int jumlahMurid = scan.nextInt();

        // Membuat array untuk menampung nama-nama murid
        String[] namaMurid = new String[jumlahMurid];

        /*
        Melakukan perulangan untuk memasukkan nama murid disetiap
```

```
        elemen array satu per satu
        */
        for(int i = 0; i < jumlahMurid; i++)
        {
            System.out.print("Nama murid ke-" + (i + 1) + " : ");
            namaMurid[i] = scan.next();
        }

        System.out.println("Nama seluruh murid: ");

        /*
        Melakukan perulangan untuk menampilakn nama murid disetiap
        elemen array satu per satu
        */
        for (String nama : namaMurid)
        {
            System.out.print(nama + " ");
        }
    }
}
```

Output Program (DemoArrayString.java)

```
Masukkan jumlah murid: 3
Nama murid ke-1 : Budi
Nama murid ke-2 : Raju
Nama murid ke-3 : Riki
Nama seluruh murid:
Budi Raju Riki
```

2.4.3 *Array* Multi Dimensi

Dalam Java kita bisa membuat *array* dengan dimensi lebih dari satu. Batas maksimal dimensi untuk *array* pada Java adalah sebanyak 255 dimensi. Nilai dalam *array* multi dimensi disimpan dalam format baris dan kolom. Berikut merupakan bentuk umum dari *array* multi dimensi.

```
tipeData[dimensiKe1][dimensiKe2][dimensiKe3]....[dimensiKeN] namaArray =
              new    tipeData[ukuranDimensi1][ukuranDimensi2][ukuranDimensi3]....
[ukuranDimensiN];
```

Array Dua Dimensi

Array dua dimensi adalah bentuk paling sederhana dari *array* multi dimensi. Berikut merupakan potongan kode untuk membuat *array* dua dimensi.

```
// Deklarasi array dua dimensi
int[][] arrayDuaDimensi;

// Instansiasi array dua dimensi
arrayDuaDimensi = new int[2][3];

// Inisialisasi nilai pada baris 0 kolom 0
arrayDuaDimensi[0][0] = 10;
// Inisialisasi nilai pada baris 0 kolom 1
arrayDuaDimensi[0][1] = 20;
// Inisialisasi nilai pada baris 0 kolom 2
arrayDuaDimensi[0][2] = 30;
```

Dari potongan kode diatas kita telah membuat array *dua* dimensi yang bertipe data `int` pada *statement* `int[][] arrayDuaDimensi;`. Dengan adanya tanda dua kurung siku setelah `int`, menandakan bahwa kita akan membuat sebuah *array* dua dimensi yang bertipe data `int`.

Selanjutnya adalah memberikan ukuran pada *array* dua dimensi pada *statement* `arrayDuaDimensi = int new [2][3];`. Tanda kurung siku yang pertama berfungsi untuk menentukan jumlah baris yaitu 2. Tanda kurung siku yang kedua berfungsi untuk menentukan jumlah kolom yaitu 3.

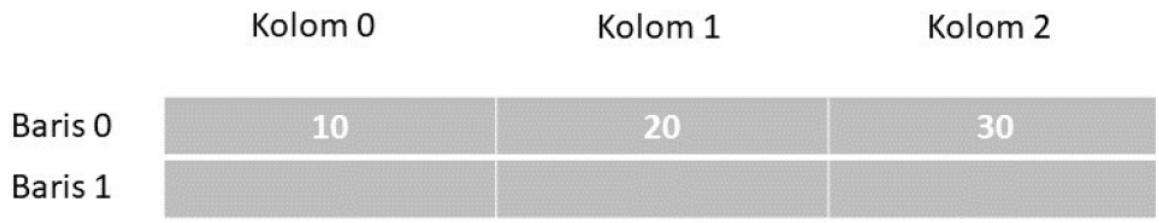
Selanjutnya adalah menginisialisasi nilai-nilai ke dalam *array* pada *statement* berikut.

```
// Inisialisasi nilai pada baris 0 kolom 0
arrayDuaDimensi[0][0] = 10;

// Inisialisasi nilai pada baris 0 kolom 1
arrayDuaDimensi[0][1] = 20;

// Inisialisasi nilai pada baris 0 kolom 2
arrayDuaDimensi[0][2] = 30;
```

Jika divisualisasikan maka `arrayDuaDimensi` akan tampak seperti berikut:



Untuk mengakses elemen pada *array* dua dimensi, harus ditentukan terlebih dahulu baris dan kolom yang ingin diakses. Sebagaai contoh perhatikan visualisasi `arrayDuaDimensi` di atas. Jika ingin mengakses elemen dari baris 0 kolom 2, maka bisa menggunakan *statement* `arrayDuaDimensi[0][2];`. Jika *statement* tersebut dieksekusi, maka akan menghasilkan nilai `30`.

Selain itu untuk mengetahui panjang dari sebuah *array* multidimensi dapat menggunakan *method* `.length`. Seperti pada *statement* berikut.

```
arrayDuaDimensi.length
```

Statement di atas jika dieksekusi maka akan menghasilkan nilai `2`. Hal tersebut dikarenakan *array* `arrayDuaDimensi` memiliki jumlah baris sebanyak 2. Berikut merupakan program untuk mencetak ukuran setiap kolom dari setiap baris dalam array 2 dimensi.

Program (PanjangArray.java)

```
public class PanjangArray
{
    public static void main(String[] args)
    {
        // Deklarasi array dua dimensi
        int[][] arrayDuaDimensi;

        // Instansiasi array dua dimensi
        arrayDuaDimensi = new int[2][3];

        // Inisialisasi nilai pada baris 0 kolom 0
        arrayDuaDimensi[0][0] = 10;
        // Inisialisasi nilai pada baris 0 kolom 1
        arrayDuaDimensi[0][1] = 20;
        // Inisialisasi nilai pada baris 0 kolom 2
        arrayDuaDimensi[0][2] = 30;

        // Mencetak ukuran baris arrayDuaDimensi
        System.out.println("Ukuran arrayDuaDimensi: " + arrayDuaDimensi.length);

        /*
         Melakukan perulangan sebanyak jumlah baris. Perulangan dilakukan
         untuk mencetak banyaknya kolom tiap baris arrayDuaDimensi
        */
        for(int index = 0; index < arrayDuaDimensi.length; index++)
        {
            System.out.println("Banyak Kolom dari baris ke-" + index +
            " adalah: " + arrayDuaDimensi[index].length);
        }
    }
}
```

Output Program (PanjangArray.java)

```
Ukuran arrayDuaDimensi: 2
Banyak Kolom dari baris ke-0 adalah: 3
Banyak Kolom dari baris ke-1 adalah: 3
```

Dari *output* di atas dapat diketahui bahwa `arrayDuaDimensi` memiliki ukuran 2 baris dan 3 kolom.

Untuk dapat mencetak seluruh elemen dari *array* dua dimensi dapat menggunakan perulangan `for` bersarang. Berikut merupakan program menampilkan *array* dua dimensi menggunakan `for` bersarang.

Program (InfoSiswa.java)

```
public class InfoSiswa
{
    public static void main(String[] args)
```

```
    {
        // Inisialisasi array dua dimensi dengan jumlah baris sebanyak 3
        String[][] infoSiswa = {
            {"Agung", "Didit", "Shana"},
            {"Budi", "Mico", "Eksel"},
            {"Rian", "Rahmat", "Farel"}};


        // Menampilkan seluruh elemen array dua dimensi dengan for bersarang

        // Perulangan pertama merupakan iterasi baris-baris pada array
       for(int baris = 0; baris < infoSiswa.length; baris++)
       {
           System.out.print("Baris ke-" + baris + ":");

           /*
           Perulangan kedua merupakan iterasi kolom yang ada
           pada baris yang sedang di iterasi
           */
           for(int kolom = 0; kolom < infoSiswa[baris].length; kolom++)
           {
               System.out.print(" " + infoSiswa[baris][kolom]);
           }

           System.out.println("");
       }
    }
}
```

Output Program (InfoSiswa.java)

```
Baris ke-0: Agung Didit Shana
Baris ke-1: Budi Mico Eksel
Baris ke-2: Rian Rahmat Farel
```

Menjumlahkan Seluruh Elemen

Penjumlahan elemen *array* dua dimensi dapat dilakukan dengan menggunakan perulangan `for` bersarang. Dalam beberapa kasus, dibutuhkan program untuk menjumlahkan elemen *array* dua dimensi seperti saat ingin membuat program menghitung keseluruhan nilai ujian. Berikut merupakan contoh program tersebut.

Program (NilaiUjian.java)

```
public class NilaiUjian
{
    public static void main(String[] args)
    {
        // Menginisialisasi array dua dimensi
        int[][] nilaiUjian = {
            {70, 100},
            {80, 75},
            {90, 80}};

        // Deklarasi dan inisialisasi variabel totalAkhir
        int totalAkhir = 0;
```

```
        // Perulangan pertama merupakan iterasi baris pada array nilaiUjian
        for(int baris = 0; baris < nilaiUjian.length; baris++)
        {
            // Deklarasi dan inisialisasi variabel totalSementara
            int totalSementara = 0;

            /*
             Perulangan kedua merupakan iterasi kolom yang ada
             pada baris yang sedang di iterasi
            */
            for(int kolom = 0; kolom < nilaiUjian[baris].length; kolom++)
            {
                /*
                 Menjumlahkan totalSementara dengan nilai pada elemen
                 array nilaiUjian indeks-[baris][kolom]
                */
                totalSementara += nilaiUjian[baris][kolom];
            }

            // Mencetak jumlah nilai pada baris yang sedang diiterasi
            System.out.print("Jumlah pada baris ke-" + baris + ": "
                             + totalSementara);

            System.out.println("");

            // Menjumlahkan totalAkhir dengan totalSementara
            totalAkhir += totalSementara;
        }

        // Mencetak jumlah nilai akhir pada variabel totalAkhir
        System.out.println("Total Keseluruhan Nilai: " + totalAkhir);
    }
}
```

Output Program (NilaiUjian.java)

```
Jumlah pada baris ke-0: 170
Jumlah pada baris ke-1: 155
Jumlah pada baris ke-2: 170
Total Keseluruhan Nilai: 495
```

Pada program di atas, diperoleh jumlah pada baris ke-0 adalah 170 dikarenakan nilai elemen `nilaiUjian[0][0]=70` dan `nilaiUjian[0][1]=100`. Lalu pada baris ke-1 adalah 155 dikarenakan nilai elemen `nilaiUjian[1][0]=80` dan `nilaiUjian[1][1]=75`. Begitu juga pada baris ke-2, jumlah yang diperoleh adalah 170 dikarenakan nilai elemen `nilaiUjian[2][0]=90` dan `nilaiUjian[2][1]=80`. Selanjutnya untuk total keselurahan nilai diperoleh dengan menjumlahkan total baris ke-0, baris ke-1, dan baris ke-2.

Array dengan Dimensi Tiga atau Lebih

Pada bahasa pemrograman Java kita bisa membuat *array* multidimensi hingga 255 dimensi. Kita dapat membuat sebuah *array* dengan dua dimensi, tiga dimensi atau bahkan lebih. Sebagai contoh, berikut deklarasi dari array tiga dimensi:

```
double[][][] arrayTigaDimensi = new double[3][4][5];
```

Untuk mengakses *array* `arrayTigaDimensi` dapat dilakukan dengan menggunakan perulangan `for` bersarang sebanyak 3 kali. Perulangan `for` yang pertama akan melakukan iterasi sebanyak jumlah yang dimasukkan pada kurung siku pertama yaitu sebanyak 3 kali. Perulangan `for` yang kedua akan melakukan iterasi sebanyak jumlah yang dimasukkan pada kurung siku kedua yaitu sebanyak 4 kali. Perulangan `for` yang ketiga akan melakukan iterasi sebanyak jumlah yang dimasukkan pada kurung siku ketiga yaitu sebanyak 5 kali.

Lalu bagaimana dengan *array* empat dimensi atau lebih?

Sama seperti *array* tiga dimensi di atas, *array* dengan empat dimensi dapat diakses dengan menggunakan perulangan `for` bersarang sebanyak 4 kali. Begitu pula untuk *array* lima dimensi atau lebih dapat diakses dengan menggunakan perulangan `for` bersarang sebanyak jumlah dimensinya.

REFERENSI

[1] Horstmann, Cay S. 2012. *Big Java: Late Objects, 1st Edition*. United States of America: John Wiley & Sons, Inc.

[2] Gaddis, Tony. 2016. *Starting Out with Java: From Control Structures through Objects (6th Edition)*. Boston: Pearson.